APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

METHOD FOR INTEGRATION OF INTERPRETATION AND TRANSLATION IN A MICROPROCESSOR

Inventors:

Robert Bedichek
Linus Torvalds
David Keppel

Prepared by:

Stephen L. King
30 Sweetbay Road
Rancho Palos Verdes, California 90275
(310) 377-5073

Express Mail Label No. EJ738560454

METHOD FOR INTEGRATION OF INTERPRETATION AND TRANSLATION IN A MICROPROCESSOR

BACKGROUND OF THE INVENTION

5 ### Field Of The Invention

This invention relates to computer systems and, more particularly, to methods and apparatus for improving the operation of a new microprocessor adapted to execute programs designed for a processor having an instruction set different than the instruction set of the new

10 microprocessor.

History Of The Prior Art

Recently, a new microprocessor was developed which combines a simple but fast host processor (called "morph host") and software (called "code morphing software") to execute application programs designed for a

15 processor (the target processor) different than the morph host processor. The morph host processor executes the code morphing software to translate the application programs into morph host processor instructions which accomplish the purpose of the original target software. As the target instructions are translated, the new host

20 instructions are both executed and stored in a translation buffer where they may be accessed without further translation. Although the initial translation and execution of a program is slow, once translated, many of the steps normally required to execute a program in hardware are eliminated. The new microprocessor has demonstrated that a simple fast

25 processor designed to expend little power is able to execute translated

"target" instructions at a rate equivalent to that of the "target" processor for which the programs were designed.

In order to be able to execute programs designed for other processors at a rapid rate, the morph host processor includes a number of hardware enhancements. One of these enhancements is a gated store buffer which resides between the host processor and the translation buffer. A second enhancement is a set of host registers which store state of the target machine at the beginning of any sequence of target instructions being translated. Generated as sequences of morph host instructions are executed and memory stores are placed in the gated store buffer. If the morph host instructions execute without raising an exception, the target state at the beginning of the sequence of instructions is updated to the target state at the point at which the sequence completed and the memory stores are committed to memory.

If an exception occurs during the execution of the sequence of host instructions which have been translated, processing stops; and the entire operation may be returned or rolled back to the beginning of the sequence of target instructions at which known state of the target machine exists in the set of host registers. This allows rapid and accurate handling of exceptions.

The combination of the code morphing software and the enhanced host processing hardware dynamically translates sequences of target instructions into sequences of instructions of a host instruction set which may be reused without being translated again. Moreover, the new processor also optimizes the translated instructions during and after the

initial translation. For example, sequences of host instructions translated from the target program may be reordered, rescheduled, and optimized in other manners to provide code which executes rapidly. Optimized sequences of translated instructions may often be linked with other translated and optimized sequences of instructions so that the process may further optimized as the instructions continue to be executed. The new processor is described in detail in U. S. patent 5,832,205, Improved Memory Control System For Microprocessor, issued November 3, 1998, to E. Kelly et al., and assigned to the assignee of the present invention.

One difficulty which has limited the speed of operation of the improved microprocessor has been that many instructions being translated and stored for reuse are reused only infrequently if at all. Because the translation process is time consuming, the average time for execution of all translated instructions is lowered by translating these little used instructions. This is especially a problem where the translated sequences have been linked to other translated sequences and significantly optimized.

In addition to the time taken to translate and optimize sequences of instructions, each translation requires storage. If each translated sequence continues to be stored in the translation buffer, an inordinate amount of storage is ultimately required.

It is desirable to increase the speed of execution of instructions by the new microprocessor while reducing the storage required for translated instructions.

Summary Of The Invention

It is, therefore, an object of the present invention to increase the speed of execution of instructions by the new microprocessor while reducing the storage required for translated instructions.

This and other objects of the present invention are realized by a method for executing a target application on a host processor including the steps of translating each target instruction being to be executed into host instructions, storing the translated host instructions, executing the translated host instructions, responding to an exception during execution of a translated instruction by rolling back to a point in execution at which correct state of a target processor is known, and interpreting each target instruction in order from the point in execution at which correct state of a target processor is known.

These and other objects and features of the invention will be better understood by reference to the detailed description which follows taken together with the drawings in which like elements are referred to by like designations throughout the several views.

Brief Description Of The Drawings

Figure 1 is a logical diagram of a microprocessor designed in accordance with the present invention running an application designed for a different microprocessor.

Figure 2 is a block diagram illustrating the microprocessor shown in Figure 1.

Figure 3 illustrates the operation of a main translation loop of the microprocessor of Figure 1.

Figure 4 is a flow chart illustrating a method practiced by a microprocessor designed in accordance with the present invention.

5   Detailed Description

The new microprocessor 10 (shown in Figure 1) combines an enhanced hardware processing portion 11 (sometimes referred to as a "morph host" in this specification) which is much simpler than typical state of the art microprocessors and an emulating software portion 13 (referred to as

10   "code morphing software" in this specification). The two portions function together as a microprocessor with advanced capabilities. The morph host 11 includes hardware enhancements to assist in providing state of a target computer immediately when an exception or error occurs, while the code morphing software 13 includes software which

15   translates the instructions of a target program into morph host instructions, optimizes the translated instructions, and responds to exceptions and errors by providing correct target state at a point at which the translation is known to be correct so that correct retranslations can occur from that point.

20   Figure 1 shows morph host hardware designed in accordance with the present invention executing a target application program 14 which is designed to run on a CISC processor such as a X86 processor. The target application and the target operating are executed on the host hardware with the assistance of the code morphing software 13.

The code morphing software 13 of the microprocessor 10 includes a translator portion 17 which decodes the instructions of the target application, converts those target instructions on the fly (dynamically) to host instructions capable of execution by the morph host, optimizes the operations required by the target instructions for execution by the morph host, reorders and schedules the primitive instructions into host instructions (a translation) for the morph host, and executes the host instructions.

In order to accelerate the operation of the new microprocessor, the code morphing software includes a translation buffer 18 as is illustrated in Figure 1. The translation buffer 18 is used to store the host instructions which embody each completed translation of the target instructions once the individual target instructions have been translated by the morph host 11. The translated host instructions may thereafter be recalled for execution whenever the operations required by the target instruction or instructions are required.

Figure 3 illustrates the operation of a main (translation) loop of the code morphing software. In a typical operation, the code morphing software of the microprocessor, when furnished the address of a target instruction, first determines whether the target instruction at the target address has been translated. If the target instruction has not been translated, it and subsequent target instructions are fetched, decoded, translated, and then (possibly) optimized, reordered, and rescheduled into a new host translation, and stored in the translation buffer by the translator. Control is then transferred to cause execution of the translation by the enhanced morph host hardware to resume.

When the particular target instruction sequence is next encountered in running the application, the host translation will be found in the translation buffer and immediately executed without the necessity of translating, optimizing, reordering, or rescheduling. Since the translation for a target instruction will be found by the morph host in the translation buffer, the myriad of steps required by the typical target processor each time it executes any instruction are eliminated. This drastically reduces the work required for executing the instructions and increases the speed of the new microprocessor.

The morph host includes hardware enhancements especially adapted to allow the acceleration techniques provided by the code morphing software to be utilized efficiently over a much broader range of instructions. Many of these hardware enhancements are used to overcome the inability of prior art techniques to handle with decent performance exceptions generated during the execution of a target program. Exceptions require that the correct target state be available at the time the exception occurs for proper execution of the exception and the instructions which follow.

In order to overcome these limitations, the enhanced morph host 11 of the new processor (shown in block form in Figure 2) incorporates a gated store buffer 50 and a large plurality of additional processor registers 40. The gated store buffer 50 stores working memory state changes on an "uncommitted" side of a hardware "gate" and official memory state changes on a "committed" side of the hardware gate where these committed stores are able to "drain" to main memory. An operation called "commit" transfers stores from the uncommitted side of the gate to

the committed side of the gate once execution of the translated code occurs without an exception. Some of the additional registers (called target registers) are used to hold the official state of the target processor at a last known correct condition; others are used as working registers. The target registers are connected to their working register equivalents through a dedicated interface that allows a commit operation to quickly transfer the content of all working registers to official target registers and allows an operation called "rollback" to quickly transfer the content of all official target registers back to their working register equivalents. The additional official registers and the gated store buffer allow the state of memory and the state of the target registers to be updated together once one or a group of target instructions have been translated and run without error.

If the primitive host instructions making up a translation of a series of target instructions are run by the host processor without generating exceptions, then the working memory stores and working register state generated by those instructions are transferred to official memory and to the official target registers. However, if an exception occurs when processing the translated host instructions at a point which is not on the boundary of a sequence of target instructions being translated, the original state in the target registers at the last update (or commit) may be recalled to the working registers and uncommitted memory stores in the gated store buffer may be discarded. Then, the target instructions causing the target exception may be retranslated one at a time as they would be executed by a target microprocessor and the translated code executed in serial sequence. As each target instruction is correctly

executed without error, the state of the target registers may be updated; and the data in the store buffer gated to memory. Then, when the target exception occurs again in running the host instructions, the correct state of the target computer is held by the target registers of the morph host and memory; and the exception may be correctly handled without delay.

In addition to simply translating the instructions, optimizing, reordering, rescheduling, storing, and executing each translation so that it may be rerun whenever that set of instructions needs to be executed, the translator also links the different translations to eliminate in almost all cases a return to the main loop of the translation process. Eventually, the main loop references in the branch instructions of host instructions are almost completely eliminated. When this condition is reached, the time required to fetch target instructions, decode target instructions, fetch the primitive instructions which make up the target instructions, optimize those primitive operations, reorder the primitive operations, and reschedule those primitive operations before running any host instruction is eliminated. Moreover, in contrast to all prior art microprocessors, long sequences of translated instructions exist which may be further optimized to increase the speed of execution.

A problem which has occurred with the new processor relates to those instructions of the target application which are seldom executed. For example, instructions required to initiate operation of a particular application are often executed only when the application is first called; and instructions required to terminate operation of an application are often executed only when the program is actually terminated. However, the new processor typically treats all instructions in the same manner. It

decodes a target instruction, fetches the primitive host instructions which carry out the function for which the target instruction is designed, proceeds through an extensive process of optimizing, and then stores the translated and optimized instruction in the translation cache. As the

5   operation of the new processor proceeds, the translated instructions are linked to one another and further optimized; and the translation of the linked instructions is stored in the translation buffer. Ultimately, large blocks of translated instructions are stored as super-blocks of host instructions. When an exception occurs during execution of a particular

10 translated instruction or linked set of instructions, the new processor goes through the process of rolling back to the last correct state of the target processor and then provides a single step translation of the target instructions from the point of the last correct state to the point at which the exception again occurs. In prior embodiments of the new processor,

15 this translation is also stored in the translation buffer.

Although this process creates code which allows rapid execution, the process has a number of effects which limit the overall speed attainable and may cause other undesirable effects. First, if a sequence of instructions is to be run but once or a few times, the time required to

20 accomplish optimizing may be significantly greater than the time needed to execute a step-by-step translation of the initial target instructions. This is especially true where the optimization accomplished includes linking translated sequences to one another. This overhead of the optimization tends to lower the average speed of the new processor.

25 Second, the process requires a substantial amount of storage capacity for translated instructions. Many times a number of different translations

exist for the same set of target instructions because the sequences were entered from different branches. Once stored, the translated instructions occupy this storage until removed for some affirmative reason.

To overcome these problems, the new processor utilizes as a part of the code morphing software, an interpreter which accomplishes step-by-step execution of target instructions. Such an interpreter could be stored as a part of host memory illustrated in Figure 2. Although there are many possible embodiments, an interpreter is essentially a program that fetches a target instruction, decodes the instruction, provides a host process to accomplish the purpose of the target instruction, and executes the host process. When it finishes interpreting one target instruction and executing the host process to carry out the result commanded by the target instruction, the interpreter precedes to the next target instruction. This process essentially single steps through the target instructions. As each target instruction is interpreted, the state of the target processor is brought up to date. The interpretation process, in contrast to the translation process, does not store host instructions in the translation buffer for future execution. The interpreter continues the process for the remainder of the sequence of target instructions.

An interpreter offers a number of advantages that are useful in certain situations. Because an interpreter causes the execution of host processes intended to carry out the purpose of each target instruction, it does not involve the complicated steps necessary to translate a sequence of target instructions. Since interpreted host processes typically are not

stored in the translation cache, linking and the further optimizations available after linking need not be carried out.

Not only does the use of an interpreter eliminate the need to optimize instructions which are not used during execution of the application and thereby increase the speed of operation, it also reduces the need for storage in the translation buffer and eliminates the need to discard many translated instructions. Interpretation may in fact be quite rapid as contrasted to translation and optimization for instructions which are little used during the execution of an application. Thus, a sequence of instructions which runs only once might be better and more rapidly handled by simply interpreting and never translating the sequence. Thus, for such instructions it may be desirable to utilize the interpreter instead of the translator software.

In order to make use of these advantages, the new processor includes apparatus and a method illustrated in Figure 4 for running the interpreter whenever a sequence of target instructions is first encountered. The interpreter software may be associated with a counter which keeps track of the number of times sequences of instructions are executed. The interpreter may be run each time the sequence is encountered until it has been executed some number of times without generating an exception. When the particular sequence of target instructions has been interpreted some selected number of times, the code morphing software switches the process from the interpreter to the translation, optimization, and storage process. When this occurs, the sequence will have been executed a sufficient number of times that it is probable that execution of the sequence will reoccur; and a stored

optimized translation will provide significantly faster execution of the application as a whole from that point on.

When the code morphing software switches to the translation process, the translation is optimized and stored in the translation cache. Thereafter, that translation may be further optimized and linked to other translations so that the very high speeds of execution realized from such processes may be obtained.

An especially useful embodiment records data relating to the number of times a target instruction is executed by the interpreter only at points at which branches occur in the instructions. The interpreter single steps through the various target instructions until a branch occurs. When a branch instruction occurs, statistics regarding that particular branch instruction (the instruction with the particular memory address) are recorded. Since all of the target instructions from the beginning of a sequence until the branch will simply be executed in sequential order, no record need be kept until the point of the branch.

Moreover, if the interpreter is utilized to collect statistics in addition to the number of times a particular target instruction has been executed, additional significant advantages may be obtained. For example, if a target instruction includes a branch, the address of the instruction to which it branches may be recorded along with the number of times the branch has been executed. Then, when a number of sequential target instructions are executed by the interpreter, a history of branching and branch addresses will have been established. From this, the likelihood of a particular branch operation taking place may be determined. These

statistics may be utilized to guide super-block formation. By utilizing these statistics, a particular sequence of instructions may be speculatively considered to be a super-block after being executed a significant number of times. After being interpreted for the selected

5 number of times, the sequence may be translated, optimized, linked through the various branches without the necessity to go through a separate linking operation, and stored as such in the translation cache. If the speculation turns out to be true, then significant time is saved in processing the instructions. If not, the operation causes an exception

10 which returns the code morphing software to the interpreter.

It has been discovered that, in addition to handling the generation of host instructions some initial number of times when a sequence of target instructions is first encountered, the interpreter may also be used advantageously when a translated sequence of instructions encounters

15 an exception. In accordance with the present invention, whenever the new processor encounters a target exception while executing any sequence of host instructions translated from a sequence of target instructions, the code morphing software causes a rollback to occur to the last known correct state of the target processor. Then, the

20 interpreter portion of the code morphing software is utilized rather than the translator portion to execute the sequence of instructions. The interpreter proceeds to interpret the target instructions to the point at which the exception occurred.

The interpreter carries out each individual one of the target instructions

25 in the sequence on a step by step basis. The interpreter fetches a target instruction, decodes the instruction, provides a host process to

accomplish the purpose of the target instruction, and executes the host process. When it finishes interpreting one target instruction and executing the host process to carry out the result commanded by the target instruction, the interpreter proceeds to the next target instruction. As each target instruction is interpreted and executed, the state of the target processor is brought up to date. The interpreter continues this process for the remainder of the sequence of target instructions until the exception again occurs. Since target state is brought up to date as each target instruction is interpreted, the state is correct at that point to correctly handle the exception.

The interpreter handles exceptions as well as the translator but offers many additional benefits. Because the interpretation process is simple, the process of determining the point of occurrence of a target exception is significantly faster than the determination of such a point when carried out by the translation process which goes through the above-described translation and optimization process and then stores host instructions in the translation buffer.

The use of the interpreter to handle the process of determining the state of the target processor at the point of a target exception eliminates the need to translate and store the host instructions used to determine that state.

By utilizing the combination of the interpreter and the optimizing translator which functions as a dynamic compiler of sequences of translated instructions to handle exceptions generated during execution of translated sequences of instructions, the code morphing software

significantly enhances the operations of the new processor. The use of the interpreter to handle exceptions has the same useful effects as using a translator for this purpose while speeding operations and reducing storage requirements.

5   Although the present invention has been described in terms of a preferred embodiment, it will be appreciated that various modifications and alterations might be made by those skilled in the art without departing from the spirit and scope of the invention. The invention should therefore be measured in terms of the claims which follow.

10   <u>What Is Claimed Is</u>: